Laval University and Lakehead University Experiments at TREC 2015 Contextual Suggestion Track

Jian Mo[1] Luc Lamontagne[1] Richard Khoury[2]
[1] Department of Computer Science and Software Engineering, Laval University
[2] Department of Software Engineering, Lakehead University

***Abstract*—In this paper we describe our effort on TREC Contextual Suggestion Track. We present a recommendation system that built upon ElasticSearch along with a machine learning re-ranking model. We utilize real world users' opinion as well as other information to build user profiles. With profile information, we then construct customized ElasticSearch queries to search on various fields. After that, a learning to rank regressor is implemented to give better ranking results. Track results of our submitted runs show the effectiveness of the system.**

Keywords—ElasticSearch, Boosting Query, Recommendation System, Point-wise re-ranking

I. INTRODUCTION

In the Contextual Suggestion Track, participants were asked to develop a system that is able to make suggestions for a particular person with a particular context. The recommendations are contextual as they are based on user's location, profile, and preferences. [1]

Two separate tasks were available in this year competition: live experiment and batch experiment. For the live experiment, each group is required to set up a live server and to respond to live requests in a short time. For batch experiments, several candidate suggestions are provided for each group of participants to re-rank the results and a final precision at 5 is calculated. Unlike last year, teams are no longer asked for a description generation. Thus the overall goal of the competition is to generate good suggestions from over 10,000 candidate attractions per city.

In this paper we present our research group's first attempt at developing a recommendation system to solve the challenge presented in the contextual suggestion task.

We participated in both tasks of this track. The fundamental approach we adopt consists in selecting similar documents to those the user likes that are highly rated by other users, combined with a re-ranking algorithm trained to predict the user's preferences. For the live experiment, we combined an ElasticSearch engine together with a SVM regressor on a virtual server in order to respond quickly to recommendation requests. For the batch experiment, we optimized our recommendation model to filter categories to mimic the user's set of preferences. We use Precision at 5 to present the results obtained for the evaluation of our system configurations. And our final P@5 results indicate that a precision of 46% can be reached with both configurations.

II. BACKGROUND

In TREC 2014 [3], the most common approach were using the language modeling framework. Most teams used positive, neutral and negative profiles in personalization of the suggestion candidates.

We also adopted this approach and developed a new profile modeling method inspired by Yang and Fang [2] in which they utilized the comments and ratings of third party users to sum up our users' opinion. Their intuition is that users with similar ratings of a place share something in common of why they like or dislike the place and our user also shares ideas with the third party users.

III. OUR APPROACH

A. System Framework

To perform our experiments, we relied on the following components:

1. Information gathering:
   a) Crawlers
2. Online Ranking Model
   a) User profile modeling
   b) ElasticSearch
   c) Customized Queries
   d) Re-ranking regressor
3. Offline re-ranking methods
   a) Category filter

The entire framework is illustrated in Figure 1 and each component is described in detail in the following sections.

B. System Overview

Figure 1 showed the overview of our system. The data are downloaded synchronously into two databases MySQL and ElasticSearch. We query ElasticSearch for candidate suggestions while get training data from MySQL for MySQL has a better API for data retrieving though ElasticSearch also has all the data we need.

Then a trained regressor will score each candidate suggestion to get a new rank. We select the top 5 suggestions as final results. In the batch run, the category filter will select final results according to their categories to maintain diversity.

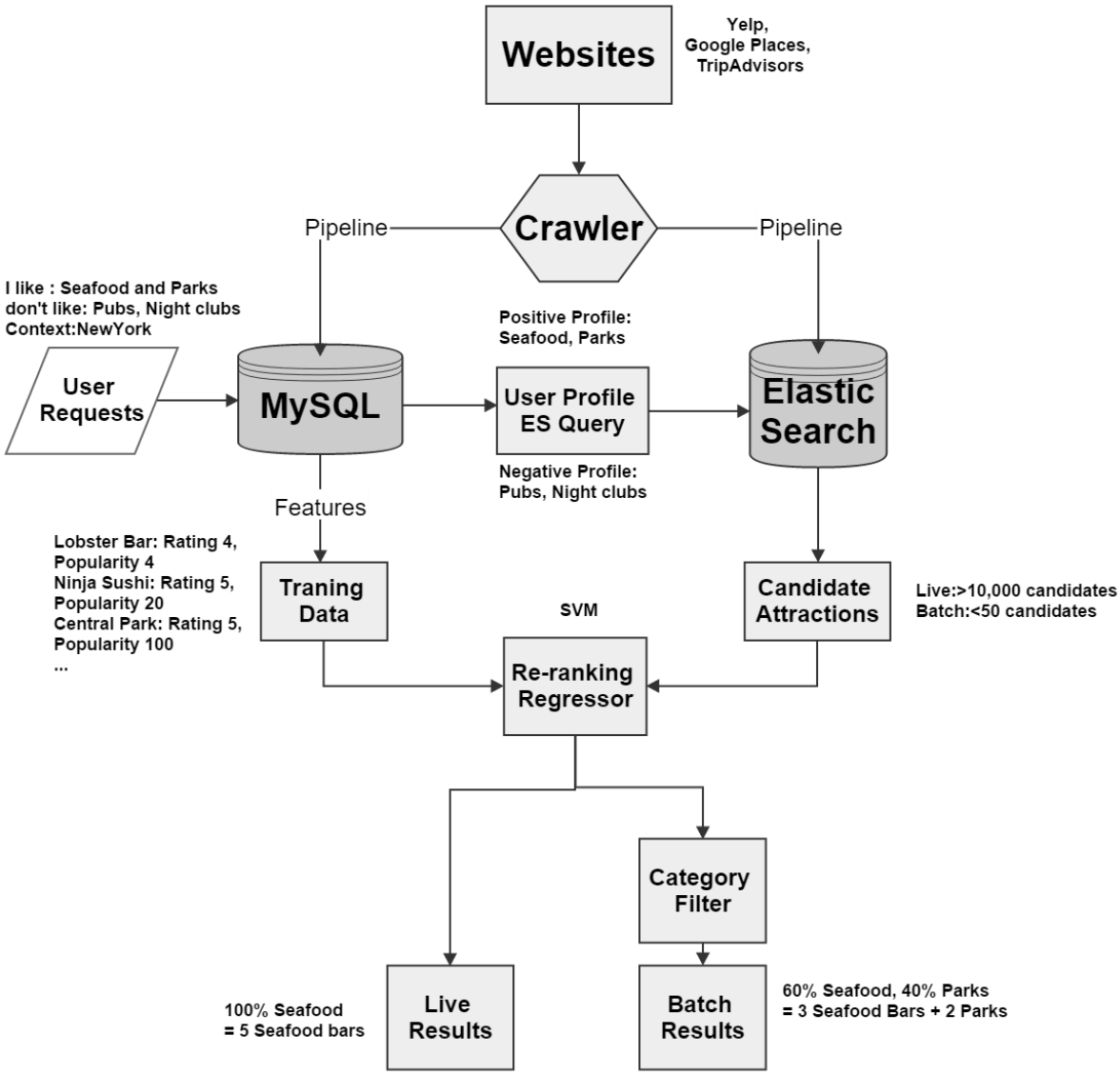

Figure 1: Overall framework for contextual recommendation

IV. Information Gathering

Over 10,000 candidate suggestions were available per context city and 273 cities in total there are as part of the challenge. The first responsibility of our system was to gather information about each of these candidate suggestions. We analysed several open-web service providers, namely Yelp, Google Places, and TripAdvisor. The last two sites provide an API that can be used to access the data source. Yelp's API is more limited, and we had to implement a custom crawler to retrieve the data from that site.

The crawler extracts useful information such as *rating*, *open hours*, *price*, *review count*, *category* for each candidate attraction from the webpages. Business information such as *has Wi-Fi*, *has parking lot*, *smoking allowed* are also extracted and stored into the database.

User reviews were often too numerous to download entirely. Hence we just get the most popular reviews of each attraction, i.e. the first page reviews. Positive and negative reviews are downloaded separately and stored in two different database tables.

Since our model has two distinct databases, we utilized the pipeline module of Scrapy to auto-sync the downloaded data between MySQL and ElasticSearch as the crawler push data to the two simultaneously.

`

V. ONLINE RANKING MODEL

A. Modeling of User Profile

While Yang and Fang [2] only used reviews as their profile data, we exploit all the data available in the database including *reviews*, *categories*, *business information*, *tags* and *other info* as our query field. We create the user's positive profile by merging the positive information from all the examples our user likes, and likewise build the negative profile by merging the negative information from the examples our user dislikes. The intuition is that the preferences of one user are reflected by the attractions he/she likes and dislikes. Consequently, we can compare the user profiles with every candidate suggestion in the database and rank them by similarity. For instance, if one candidate suggestion has many elements in common with the positive user profile, this candidate obtains a higher ranking score. In contrast, if it is very similar to the negative user profile, its ranking score is penalized and very low.

According to the task defined for this track, one user might provide 6 different rating:

4: Strongly interested
3: Interested
2: Neutral
1: Disinterested
0: Strongly disinterested
-1: Website didn't load or no rating given

We selected rating 4 and 3 as positive and 1 and 0 as negative. Ratings 2 and -1 were taken as neutral and simply ignored.

Formally, the user profile can be expressed as:

$$Profile_{Pos} = \bigcup REP(Pes_{ik})$$

$$Profile_{Neg} = \bigcup REP(Nes_{ik})$$

Where Pes_i is positive example suggestion i, and Pes_{ik} is element k (categories, tags, positive reviews, business info) of this positive suggestion. $REP(Pes_{ik})$ defines a special representation or form of the element. For example, the review texts of third party users (which can be quite numerous and quite lengthy) are represented by the 50 most frequently used keywords in the merged set of reviews. After being properly processed, all example suggestions from one user are merged together to form the user profile. At the end of the modeling process, we get four different positive profile elements: positive categories, positive business info, positive tags, and positive reviews. The same steps are performed to build the negative profile. For instance, one possible positive category is a long string like "parks, restaurant, outdoor sports, parks, playground…" which sums up all the categories user has rated positive.

B. Query Formulation

Once the user profile is built, we formulate a customized query to search our ElasticSearch database. For example, suppose the user likes Mexican food, dislikes Japanese food, appreciates Wi-Fi and parking lot and hates smoking. Part of our query could be formulated as the json structure in Figure 2. As can be seen, we use a bool boosting query to wrap up the elements of the user profile into one query, which can then be sent to ElasticSearch to retrieve relevant new attractions and similarity scores. These similarity scores will be used by our ranking function.

```
   "bool": {
   "should": [
   {
   "boosting": {
     "positive": {
       "match": {
         "category": "Mexican
food"
       }
     },
     "negative": {
       "match": {
         "category": "Japanese
food"
       }
     },
     "negative_boost": 0.3
     }
     },
   {
   "boosting": {
     "positive": {
       "match": {
         "business_info": "has
wifi, good for kids"
       }
     },
     "negative": {
       "match": {
         "business_info": "smok-
ing"
       }
     },
     "negative_boost": 0.1
   }
   }
```

Figure 2: Json query example.

C. ranking Methods

The similarity score provided by elastic search in the previous step is sufficient to give satisfactory

`

ranking results. However, to improve the performance of our system, we combine this similarity score with other features such as *category*, *popularity*, *and rating* and utilize a learning algorithm to compute a final ranking score for each candidate. We compute a personal regressor for every user according to his/her own preference instead of building a global regressor that predicts common interests shared by all people.

During our experiments, we tried several learning methods including Linear Regression, SVM and LambdaMart [4] with different combination of features. We used a 5-fold cross validation training algorithm on last year's dataset to evaluate each configuration. We found that Linear Regression and Lambda Mart perform poorly in this case, because the size of the training data per user is small (less than 50 samples). On the other hand, we observed that SVM applied on all the features we gathered during profile modeling achieves the best ranking performance.

VI. Re-ranking methods in Batch mode

Our first batch run used the same model as the live run using the SVM-based ranking scheme we described in Section V. For the second batch run, we added a category filter that we developed in order to bias the set of recommendations returned to mimic the diversity of preferences of the user. Our intuition for this filter is that there must be a diminishing return to suggestions of one same type, reflecting the user's decreasing interest in visiting several similar attractions. Consequently, we decided to add diversity to our recommendations to avoid having too many recommendations in one category and to insure we have recommendations in the major categories the user has shown interest in.

This category filter first calculates the distribution of categories in the user positive profile by summing up the counts of every positive category and then divide the total counts to get the proportion of each category. When generating a set of recommendations for a new city, attractions are selected or rejected so that the set will have a similar proportion to the user's profile. For example, if the user profile has a distribution composed of 40% restaurants and 60% museums and the system is designed to return 50 recommendations, then it will return the top 20 restaurants and top 30 museums ranked similarity scores.

VII. Results and Analysis

A. Evaluation Metric

An attraction is considered relevant for P@5 if it has a geographical relevance of 1 or 2 and if the user reported that both the description and document were found to be interesting (3) or strongly interesting (4). A P@5 score for a particular topic (a profile-context pair) is determined by how many of the top 5 ranked attractions are relevant, divided by 5. [1]

B. Submitted Runs

Three runs were submitted to the competition:

- LavalIVA-run1, our live run,
- LavalIVA_1 is batch run 1, using the same model as the live run,
- LavalIVA_2 is batch run 2, which applies the category filter

Table 1: TREC CS 2015 results

Run	P@5
LavalIVA-batch	0.2611
LavalIVA_1	0.4645
LavalIVA_2	0.4616

Table 1 shows our final result [1]. As showed from the table, the precision of the first batch run (LavalIVA_1) is much higher than those obtained for the live run (LavalIVA-batch). The difference is not the ranking model or data, which were identical in both versions of the system, but rather that, unfortunately, our live server failed to respond to multiple requests made in the first two days of the evaluation. Furthermore, a few blank responses were provided during the two week period, which is due to a lack of robustness in our online service.

The track medium P@5 of all batch runs is 0.4946 while the highest is 0.5858. After analyzing our system on qrels, we found two major problems of our system in batch run test. The first problem lies in our data richness. Since we dropped some of the candidates' data source like Wikipedia and some other websites, we don't have all the data of candidates in the batch run as the candidates were provided by other participates system generated in the live run. Thus our system had some blank user profiles and returned false results. The second problem is our system has difficulty in dealing with user who has few positive ratings since our regressors are trained individually on single user so that when user has few positive data our regressors could not be trained properly. This problem is also found in the live run phase. Our system often obtains good score when the user provides sufficient positive feedbacks.

We also observe that the application of a diversity filter slightly lowers the quality of the recom-

`

mendations on P@5 standard. This seems to indicate that the best recommendations may often fall into the same category for some users.

VIII. Conclusion

In this paper, we have presented our contextual suggestion system. We have described its major components: building new user profile models, retrieving suggestions using ElasticSearch, and reranking the results using a regressor and a category filter. While our batch runs demonstrated a solid performance, the system used for the live run requires improvements to improve its robustness. For future improvements, we plan to replace the private regressors with a global regressor to deal with the problem of data sparseness in the user profiles and also enrich our data source to obtain better result in batch run.

IX. References

[1] A. Dean-Hall, C. Clarke, J. Kamps, P. Thomas and E. Voorhees. Overview of the TREC 2015 Contextual Suggestion Track. In Proceedings of TREC'15, 2015.

[2] P. Yang and H. Fang. An opinion-aware approach to contextual suggestion. In Proceedings of TREC'13, 2013.

[3] A. Dean-Hall, C. Clarke, J. Kamps, P. Thomas, N. Simone, and E. Voorhees. Overview of the TREC 2014 contextual suggestion track. In Proceedings of TREC'14, 2014

[4] C. J. C. Burges. From RankNet to LambdaRank to LambdaMART: An overview. Technical report, Microsoft Research, 2010.

`